Contents

Authentication Policy

13 minute read ✔ page test

This task covers the primary activities you might need to perform when enabling, configuring, and using Istio authentication policies. Find out more about the underlying concepts in the authentication overview.

Before you begin

- Understand Istio authentication policy and related mutual TLS authentication concepts.
- Install Istio on a Kubernetes cluster with the `default` configuration profile, as described in installation steps.

```
$ istioctl install --set profile=default
```

Setup

Our examples use two namespaces `foo` and `bar`, with two services, `httpbin` and `sleep`, both running with an Envoy proxy. We also use second instances of `httpbin` and `sleep` running without the sidecar in the `legacy` namespace. If you'd like to use the same examples when trying the tasks, run the following:

```
$ kubectl create ns foo
$ kubectl apply -f <(istioctl kube-inject -f @samples/httpbin/ht
tpbin.yaml@) -n foo
$ kubectl apply -f <(istioctl kube-inject -f @samples/sleep/slee
p.yaml@) -n foo
$ kubectl create ns bar
$ kubectl apply -f <(istioctl kube-inject -f @samples/httpbin/ht
tpbin.yaml@) -n bar
$ kubectl apply -f <(istioctl kube-inject -f @samples/sleep/slee
p.yaml@) -n bar
$ kubectl create ns legacy
$ kubectl apply -f @samples/httpbin/httpbin.yaml@ -n legacy
$ kubectl apply -f @samples/sleep/sleep.yaml@ -n legacy
```

You can verify setup by sending an HTTP request with `curl` from any `sleep` pod in the namespace `foo`, `bar` or `legacy` to either `httpbin.foo`, `httpbin.bar` or

`httpbin.legacy`. All requests should succeed with HTTP code 200.

For example, here is a command to check `sleep.bar` to `httpbin.foo` reachability:

```
$ kubectl exec "$(kubectl get pod -l app=sleep -n bar -o jsonpath
h={.items..metadata.name})" -c sleep -n bar -- curl http://httpb
in.foo:8000/ip -s -o /dev/null -w "%{http_code}\n"
200
```

This one-liner command conveniently iterates through all reachability combinations:

```
$ for from in "foo" "bar" "legacy"; do for to in "foo" "bar" "le
gacy"; do kubectl exec "$(kubectl get pod -l app=sleep -n ${from
} -o jsonpath={.items..metadata.name})" -c sleep -n ${from} -- c
url -s "http://httpbin.${to}:8000/ip" -s -o /dev/null -w "sleep.
${from} to httpbin.${to}: %{http_code}\n"; done; done
sleep.foo to httpbin.foo: 200
sleep.foo to httpbin.bar: 200
sleep.foo to httpbin.legacy: 200
sleep.bar to httpbin.foo: 200
sleep.bar to httpbin.bar: 200
sleep.bar to httpbin.legacy: 200
sleep.legacy to httpbin.foo: 200
sleep.legacy to httpbin.bar: 200
sleep.legacy to httpbin.legacy: 200
```

Verify there is no peer authentication policy in the system with the following command:

```
$ kubectl get peerauthentication --all-namespaces
No resources found
```

Last but not least, verify that there are no destination rules that apply on the example services. You can do this by checking the `host:` value of existing destination rules and make sure they do not match. For example:

```
$ kubectl get destinationrules.networking.istio.io --all-namespaces -o yaml | grep "host:"
```

> Depending on the version of Istio, you may see destination rules for hosts other than

those shown. However, there should be none with hosts in the `foo`, `bar` and `legacy` namespace, nor is the match-all wildcard `*`

Auto mutual TLS

By default, Istio tracks the server workloads migrated to Istio proxies, and configures client proxies to send mutual TLS traffic to those workloads automatically, and to send plain text traffic to workloads without

sidecars.

Thus, all traffic between workloads with proxies uses mutual TLS, without you doing anything. For example, take the response from a request to `httpbin/header`. When using mutual TLS, the proxy injects the `X-Forwarded-Client-Cert` header to the upstream request to the backend. That header's presence is evidence that mutual TLS is used. For example:

```
$ kubectl exec "$(kubectl get pod -l app=sleep -n foo -o jsonpat
h={.items..metadata.name})" -c sleep -n foo -- curl -s http://ht
tpbin.foo:8000/headers -s | grep X-Forwarded-Client-Cert | sed '
s/Hash=[a-z0-9]*;/Hash=<redacted>;/'
    "X-Forwarded-Client-Cert": "By=spiffe://cluster.local/ns/foo
/sa/httpbin;Hash=<redacted>;Subject=\"\";URI=spiffe://cluster.lo
cal/ns/foo/sa/sleep"
```

When the server doesn't have sidecar, the `X-Forwarded-Client-Cert` header is not there, which implies requests are in plain text.

```
$ kubectl exec "$(kubectl get pod -l app=sleep -n foo -o jsonpat
h={.items..metadata.name})" -c sleep -n foo -- curl http://httpb
in.legacy:8000/headers -s | grep X-Forwarded-Client-Cert
```

Globally enabling Istio mutual TLS in STRICT mode

While Istio automatically upgrades all traffic between the proxies and the workloads to mutual TLS, workloads can still receive plain text traffic. To prevent non-mutual TLS traffic for the whole mesh, set a mesh-wide peer authentication policy with the mutual TLS mode set to `STRICT`. The mesh-wide peer authentication policy should not have a `selector` and must be applied in the **root namespace**, for example:

```
$ kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: "default"
  namespace: "istio-system"
spec:
  mtls:
    mode: STRICT
EOF
```

The example assumes `istio-system` is the root namespace. If you used a different value during installation, replace `istio-system` with the value you used.

This peer authentication policy configures workloads to only accept requests encrypted with TLS. Since it doesn’t specify a value for the `selector` field, the policy applies to all workloads in the mesh.

Run the test command again:

```
$ for from in "foo" "bar" "legacy"; do for to in "foo" "bar" "le
gacy"; do kubectl exec "$(kubectl get pod -l app=sleep -n ${from
} -o jsonpath={.items..metadata.name})" -c sleep -n ${from} -- c
url "http://httpbin.${to}:8000/ip" -s -o /dev/null -w "sleep.${f
rom} to httpbin.${to}: %{http_code}\n"; done; done
sleep.foo to httpbin.foo: 200
sleep.foo to httpbin.bar: 200
sleep.foo to httpbin.legacy: 200
sleep.bar to httpbin.foo: 200
sleep.bar to httpbin.bar: 200
sleep.bar to httpbin.legacy: 200
sleep.legacy to httpbin.foo: 000
command terminated with exit code 56
sleep.legacy to httpbin.bar: 000
command terminated with exit code 56
sleep.legacy to httpbin.legacy: 200
```

You see requests still succeed, except for those from

the client that doesn’t have proxy, `sleep.legacy`, to the server with a proxy, `httpbin.foo` or `httpbin.bar`. This is expected because mutual TLS is now strictly required, but the workload without sidecar cannot comply.

Cleanup part 1

Remove global authentication policy and destination rules added in the session:

```
$ kubectl delete peerauthentication -n istio-system default
```

Enable mutual TLS per namespace or workload

Namespace-wide policy

To change mutual TLS for all workloads within a particular namespace, use a namespace-wide policy. The specification of the policy is the same as for a mesh-wide policy, but you specify the namespace it applies to under `metadata`. For example, the following peer authentication policy enables strict mutual TLS

for the `foo` namespace:

```
$ kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: "default"
  namespace: "foo"
spec:
  mtls:
    mode: STRICT
EOF
```

As this policy is applied on workloads in namespace `foo` only, you should see only request from client-without-sidecar (`sleep.legacy`) to `httpbin.foo` start to fail.

```
$ for from in "foo" "bar" "legacy"; do for to in "foo" "bar" "le
gacy"; do kubectl exec "$(kubectl get pod -l app=sleep -n ${from
} -o jsonpath={.items..metadata.name})" -c sleep -n ${from} -- c
url "http://httpbin.${to}:8000/ip" -s -o /dev/null -w "sleep.${f
rom} to httpbin.${to}: %{http_code}\n"; done; done
sleep.foo to httpbin.foo: 200
sleep.foo to httpbin.bar: 200
sleep.foo to httpbin.legacy: 200
sleep.bar to httpbin.foo: 200
sleep.bar to httpbin.bar: 200
sleep.bar to httpbin.legacy: 200
sleep.legacy to httpbin.foo: 000
command terminated with exit code 56
sleep.legacy to httpbin.bar: 200
sleep.legacy to httpbin.legacy: 200
```

Enable mutual TLS per

workload

To set a peer authentication policy for a specific workload, you must configure the `selector` section and specify the labels that match the desired workload. However, Istio cannot aggregate workload-level policies for outbound mutual TLS traffic to a service. Configure a destination rule to manage that behavior.

For example, the following peer authentication policy and destination rule enable strict mutual TLS for the `httpbin.bar` workload:

```
$ cat <<EOF | kubectl apply -n bar -f -
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: "httpbin"
  namespace: "bar"
spec:
  selector:
    matchLabels:
      app: httpbin
  mtls:
    mode: STRICT
EOF
```

And a destination rule:

```
$ cat <<EOF | kubectl apply -n bar -f -
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: "httpbin"
spec:
  host: "httpbin.bar.svc.cluster.local"
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
EOF
```

Again, run the probing command. As expected, request from `sleep.legacy` to `httpbin.bar` starts failing with the same reasons.

```
$ for from in "foo" "bar" "legacy"; do for to in "foo" "bar" "le
gacy"; do kubectl exec "$(kubectl get pod -l app=sleep -n ${from
} -o jsonpath={.items..metadata.name})" -c sleep -n ${from} -- c
url "http://httpbin.${to}:8000/ip" -s -o /dev/null -w "sleep.${f
rom} to httpbin.${to}: %{http_code}\n"; done; done
sleep.foo to httpbin.foo: 200
sleep.foo to httpbin.bar: 200
sleep.foo to httpbin.legacy: 200
sleep.bar to httpbin.foo: 200
sleep.bar to httpbin.bar: 200
sleep.bar to httpbin.legacy: 200
sleep.legacy to httpbin.foo: 000
command terminated with exit code 56
sleep.legacy to httpbin.bar: 000
command terminated with exit code 56
sleep.legacy to httpbin.legacy: 200
```

```
...
sleep.legacy to httpbin.bar: 000
command terminated with exit code 56
```

To refine the mutual TLS settings per port, you must configure the `portLevelMtls` section. For example, the following peer authentication policy requires mutual TLS on all ports, except port `80`:

```
$ cat <<EOF | kubectl apply -n bar -f -
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: "httpbin"
  namespace: "bar"
spec:
  selector:
    matchLabels:
      app: httpbin
  mtls:
    mode: STRICT
  portLevelMtls:
    80:
      mode: DISABLE
EOF
```

As before, you also need a destination rule:

```
$ cat <<EOF | kubectl apply -n bar -f -
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: "httpbin"
spec:
  host: httpbin.bar.svc.cluster.local
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
    portLevelSettings:
    - port:
        number: 8000
      tls:
        mode: DISABLE
EOF
```

1. The port value in the peer authentication policy is the container's port. The value the destination

rule is the service's port.

2. You can only use `portLevelMtls` if the port is bound to a service. Istio ignores it otherwise.

```
$ for from in "foo" "bar" "legacy"; do for to in "foo" "bar" "le
gacy"; do kubectl exec "$(kubectl get pod -l app=sleep -n ${from
} -o jsonpath={.items..metadata.name})" -c sleep -n ${from} -- c
url "http://httpbin.${to}:8000/ip" -s -o /dev/null -w "sleep.${f
rom} to httpbin.${to}: %{http_code}\n"; done; done
sleep.foo to httpbin.foo: 200
sleep.foo to httpbin.bar: 200
sleep.foo to httpbin.legacy: 200
sleep.bar to httpbin.foo: 200
sleep.bar to httpbin.bar: 200
sleep.bar to httpbin.legacy: 200
sleep.legacy to httpbin.foo: 000
command terminated with exit code 56
sleep.legacy to httpbin.bar: 200
sleep.legacy to httpbin.legacy: 200
```

Policy precedence

A workload-specific peer authentication policy takes precedence over a namespace-wide policy. You can test this behavior if you add a policy to disable mutual TLS for the `httpbin.foo` workload, for example. Note that you've already created a namespace-wide policy that enables mutual TLS for all services in namespace `foo` and observe that requests from `sleep.legacy` to `httpbin.foo` are failing (see above).

```
$ cat <<EOF | kubectl apply -n foo -f -
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: "overwrite-example"
  namespace: "foo"
spec:
  selector:
    matchLabels:
      app: httpbin
  mtls:
    mode: DISABLE
EOF
```

and destination rule:

```
$ cat <<EOF | kubectl apply -n foo -f -
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: "overwrite-example"
spec:
  host: httpbin.foo.svc.cluster.local
  trafficPolicy:
    tls:
      mode: DISABLE
EOF
```

Re-running the request from `sleep.legacy`, you should see a success return code again (200), confirming service-specific policy overrides the namespace-wide policy.

```
$ kubectl exec "$(kubectl get pod -l app=sleep -n legacy -o json
path={.items..metadata.name})" -c sleep -n legacy -- curl http:/
/httpbin.foo:8000/ip -s -o /dev/null -w "%{http_code}\n"
200
```

Cleanup part 2

Remove policies and destination rules created in the above steps:

```
$ kubectl delete peerauthentication default overwrite-example -n
 foo
$ kubectl delete peerauthentication httpbin -n bar
$ kubectl delete destinationrules overwrite-example -n foo
$ kubectl delete destinationrules httpbin -n bar
```

End-user authentication

To experiment with this feature, you need a valid JWT. The JWT must correspond to the JWKS endpoint you want to use for the demo. This tutorial use the test token JWT test and JWKS endpoint from the Istio

code base.

Also, for convenience, expose `httpbin.foo` via `ingressgateway` (for more details, see the `ingress task`).

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: httpbin-gateway
  namespace: foo
spec:
  selector:
    istio: ingressgateway # use Istio default gateway implementa
tion
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
EOF
```

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin
  namespace: foo
spec:
  hosts:
  - "*"
  gateways:
  - httpbin-gateway
  http:
  - route:
    - destination:
        port:
          number: 8000
        host: httpbin.foo.svc.cluster.local
EOF
```

Follow the instructions in Determining the ingress IP and ports to define the `INGRESS_HOST` and `INGRESS_PORT` environment variables.

And run a test query

```
$ curl "$INGRESS_HOST:$INGRESS_PORT/headers" -s -o /dev/null -w
"%{http_code}\n"
200
```

Now, add a request authentication policy that requires end-user JWT for the ingress gateway.

```
$ kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: "jwt-example"
  namespace: istio-system
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  jwtRules:
  - issuer: "testing@secure.istio.io"
    jwksUri: "https://raw.githubusercontent.com/istio/istio/rele
ase-1.11/security/tools/jwt/samples/jwks.json"
EOF
```

Apply the policy to the namespace of the workload it selects, `ingressgateway` in this case. The namespace

you need to specify is then `istio-system`.

If you provide a token in the authorization header, its implicitly default location, Istio validates the token using the public key set, and rejects requests if the bearer token is invalid. However, requests without tokens are accepted. To observe this behavior, retry the request without a token, with a bad token, and with a valid token:

```
$ curl "$INGRESS_HOST:$INGRESS_PORT/headers" -s -o /dev/null -w
"%{http_code}\n"
200
```

```
$ curl --header "Authorization: Bearer deadbeef" "$INGRESS_HOST:
$INGRESS_PORT/headers" -s -o /dev/null -w "%{http_code}\n"
401
```

```
$ TOKEN=$(curl https://raw.githubusercontent.com/istio/istio/rel
ease-1.11/security/tools/jwt/samples/demo.jwt -s)
$ curl --header "Authorization: Bearer $TOKEN" "$INGRESS_HOST:$I
NGRESS_PORT/headers" -s -o /dev/null -w "%{http_code}\n"
200
```

To observe other aspects of JWT validation, use the script `gen-jwt.py` to generate new tokens to test with different issuer, audiences, expiry date, etc. The script can be downloaded from the Istio repository:

```
$ wget --no-verbose https://raw.githubusercontent.com/istio/istio/release-1.11/security/tools/jwt/samples/gen-jwt.py
```

You also need the `key.pem` file:

```
$ wget --no-verbose https://raw.githubusercontent.com/istio/istio/release-1.11/security/tools/jwt/samples/key.pem
```

Download the `jwcrypto` library, if you haven't installed it on your system.

The JWT authentication has 60 seconds clock skew,

this means the JWT token will become valid 60 seconds earlier than its configured `nbf` and remain valid 60 seconds after its configured `exp`.

For example, the command below creates a token that expires in 5 seconds. As you see, Istio authenticates requests using that token successfully at first but rejects them after 65 seconds:

```
$ TOKEN=$(python3 ./gen-jwt.py ./key.pem --expire 5)
$ for i in $(seq 1 10); do curl --header "Authorization: Bearer
$TOKEN" "$INGRESS_HOST:$INGRESS_PORT/headers" -s -o /dev/null -w
 "%{http_code}\n"; sleep 10; done
200
200
200
200
200
200
200
200
401
401
401
```

You can also add a JWT policy to an ingress gateway (e.g., service `istio-ingressgateway.istio-system.svc.cluster.local`). This is often used to define a

JWT policy for all services bound to the gateway, instead of for individual services.

Require a valid token

To reject requests without valid tokens, add an authorization policy with a rule specifying a `DENY` action for requests without request principals, shown as `notRequestPrincipals: ["*"]` in the following example. Request principals are available only when valid JWT tokens are provided. The rule therefore denies requests without valid tokens.

```
$ kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: "frontend-ingress"
  namespace: istio-system
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  action: DENY
  rules:
  - from:
    - source:
        notRequestPrincipals: ["*"]
EOF
```

Retry the request without a token. The request now

fails with error code `403`:

```
$ curl "$INGRESS_HOST:$INGRESS_PORT/headers" -s -o /dev/null -w
"%{http_code}\n"
403
```

Require valid tokens per-path

To refine authorization with a token requirement per host, path, or method, change the authorization policy to only require JWT on `/headers`. When this

authorization rule takes effect, requests to `$INGRESS_HOST:$INGRESS_PORT/headers` fail with the error code `403`. Requests to all other paths succeed, for example `$INGRESS_HOST:$INGRESS_PORT/ip`.

```
$ kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: "frontend-ingress"
  namespace: istio-system
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  action: DENY
  rules:
  - from:
    - source:
        notRequestPrincipals: ["*"]
    to:
    - operation:
        paths: ["/headers"]
EOF
```

```
$ curl "$INGRESS_HOST:$INGRESS_PORT/headers" -s -o /dev/null -w
"%{http_code}\n"
403
```

```
$ curl "$INGRESS_HOST:$INGRESS_PORT/ip" -s -o /dev/null -w "%{ht
tp_code}\n"
200
```

Cleanup part 3

1. Remove authentication policy:

    ```
    $ kubectl -n istio-system delete requestauthentication jwt-
    example
    ```

2. Remove authorization policy:

```
$ kubectl -n istio-system delete authorizationpolicy fronte
nd-ingress
```

3. Remove the token generator script and key file:

```
$ rm -f ./gen-jwt.py ./key.pem
```

4. If you are not planning to explore any follow-on tasks, you can remove all resources simply by deleting test namespaces.

```
$ kubectl delete ns foo bar legacy
```